Jelaskan lebih jauh masing-masing algoritma di bawah dengan detail :

- Kinematics (object detection, pose estimation, camera calibration)
- ADRC (Active Disturbance Rejection Control)
- PID (Proportional-Integral-Derivative) control algorithms
- A* (A star) algorithm

JAWABAN

KINEMATICS (object detection, pose estimation, camera calibration)

Kinematics adalah kerangka matematika yang digunakan untuk mendefinisikan dan menghitung gerakan sistem mekanik seperti robot, tanpa memperhatikan gaya, posisi, atau kecepatan yang terlibat. Dalam konteks robotika, kinematics terbagi menjadi dua jenis utama: Forward Kinematics dan Inverse Kinematics. Forward Kinematics melakukan kalkulasi untuk menentukan posisi bagian ujung robot berdasarkan sudut sendi yang ada. Dengan kata lain, kita mengetahui sudut masing-masing sendi dan menghitung di mana posisi ujung manipulator robot akan berada. Sebaliknya, Inverse Kinematics melakukan kalkulasi untuk menentukan sudut-sudut sendi yang diperlukan agar bagian ujung robot berada pada posisi tertentu. Ini lebih kompleks karena robot harus menghitung bagaimana semua sendinya harus bergerak untuk mencapai titik target dengan tepat.

Object Detection merupakan langkah awal penting dalam robotika yang digunakan untuk mengidentifikasi dan menentukan lokasi objek dalam gambar atau video. Meskipun Object Detection dan kinematics tidak secara langsung berhubungan, keduanya bekerja secara sinergis dalam aplikasi robotika. Ketika robot harus mengambil sebuah objek, ia perlu terlebih dahulu melakukan Object Detection untuk mengetahui posisi benda tersebut di lingkungan 3D. Setelah objek berhasil diidentifikasi, algoritma kinematics akan menentukan gerakan sendi robot yang diperlukan agar robot dapat mencapai, memegang, dan memindahkan objek tersebut.

Pose Estimation adalah proses yang digunakan untuk menentukan posisi dan orientasi (sikap) suatu objek di dalam ruang tiga dimensi. Dalam hal ini, kinematics berperan penting untuk memahami bagaimana sikap atau orientasi tubuh robot, terutama pada robot yang meniru gerakan manusia. Proses ini sering disebut sebagai "kinematics chain," di mana segmen tubuh robot yang kaku dihubungkan dengan berbagai sendi. Dalam pose estimation, kinematics memungkinkan robot untuk menyesuaikan derajat tekuk sendi-sendinya sehingga dapat meniru postur atau sikap yang diinginkan.

Camera Calibration adalah tahap yang sangat penting dalam proses kinematics dan visi komputer. Kalibrasi kamera melibatkan penentuan parameter intrinsik dan ekstrinsik kamera yang digunakan untuk memetakan dunia tiga dimensi ke dalam citra dua dimensi dan sebaliknya. Parameter intrinsik meliputi panjang fokus kamera, pusat optik, dan koefisien distorsi yang membantu mengubah data dari lingkungan 3D menjadi gambar 2D. Sedangkan, parameter ekstrinsik mencakup posisi dan orientasi kamera relatif terhadap koordinat robot, yang memungkinkan konversi dari koordinat 2D ke 3D. Hasil dari camera calibration memberikan matriks transformasi yang dapat digunakan oleh algoritma kinematics untuk menghitung gerakan robot terhadap objek yang teridentifikasi di lingkungannya.

Secara keseluruhan, kinematics memainkan peran integral dalam pengembangan sistem robot yang kompleks, mulai dari mengidentifikasi objek dan mengestimasi posisinya hingga mengontrol gerakan sendi robot secara presisi. Kombinasi antara algoritma kinematics, object detection, pose estimation, dan camera calibration memastikan bahwa robot dapat berinteraksi secara efektif dengan lingkungannya, mendeteksi objek, menghitung posisi objek, dan bergerak secara tepat untuk melakukan tugas yang diinginkan.

ADRC (Active Disturbance Rejection Control)

ADRC (Active Disturbance Rejection Control) adalah tipe metode kontrol kuat nonlinier yang dirancang untuk mengatasi ketidakpastian dan gangguan pada sistem, baik internal maupun eksternal. Algoritma ini bekerja dengan menambahkan variabel tambahan dan fiktif ke dalam model sistem, yang mewakili segala sesuatu yang tidak dimasukkan oleh pengguna dalam model matematika robot. ADRC sangat efektif dalam kondisi di mana sistem memiliki dinamika yang tidak diketahui dan beragam gangguan, karena mampu beradaptasi dengan perubahan yang tidak terduga. Secara teknis, ADRC dapat dianggap sebagai penerus dari teknologi PID (Proportional-Integral-Derivative) dengan performa yang lebih unggul. Salah satu keunggulan utama ADRC dibandingkan PID adalah kemampuannya untuk mengatasi gangguan baik internal maupun eksternal, yang membuat tegangan keluaran lebih sesuai dengan nilai yang diharapkan meskipun terjadi perubahan mendadak pada tegangan masukan.

ADRC terbagi menjadi beberapa komponen utama yang bekerja bersama untuk mencapai kontrol yang optimal: Tracking Differentiator (TD), Extended State Observer (ESO), dan Nonlinear State Error Feedback (NLSEF). Tracking Differentiator (TD) digunakan untuk mengubah masukan referensi yang tiba-tiba atau berisik menjadi lintasan yang lebih halus dan teratur, sehingga mengurangi osilasi yang tidak diinginkan. Selanjutnya, Extended State Observer (ESO) memiliki peran kunci dalam mengamati dan memperkirakan gangguan yang terjadi pada sistem. ESO tidak hanya mengukur kondisi sistem saat ini, tetapi juga memprediksi gangguan yang mungkin mempengaruhi performa sistem dan menghasilkan koreksi berdasarkan hasil pengamatan tersebut. Terakhir, Nonlinear State Error Feedback (NLSEF) bertanggung jawab untuk menstabilkan sistem menggunakan umpan balik yang didasarkan pada kesalahan antara kondisi yang diinginkan dengan perkiraan kondisi dari ESO. Dengan menggunakan pendekatan nonlinier ini, NLSEF dapat memberikan respons kontrol yang lebih cepat dan lebih presisi dibandingkan metode linier tradisional.

Secara keseluruhan, ADRC merupakan metode kontrol yang jauh lebih adaptif dibandingkan PID karena mampu menyesuaikan diri dengan berbagai jenis gangguan dan perubahan kondisi dalam sistem. Algoritma ini banyak diterapkan pada sistem yang kompleks dengan dinamika yang sulit diprediksi, seperti kontrol motor listrik, robotika, dan berbagai aplikasi industri lainnya, di mana ketangguhan terhadap gangguan sangat diperlukan untuk menjaga stabilitas dan kinerja optimal.

Sumber:

https://www.youtube.com/watch?v=snH8-fpuNJA
https://www.mathworks.com/help/slcontrol/ug/active-disturbance-rejection-control.html
https://onlinelibrary.wiley.com/doi/10.1155/2023/8429922
https://www.sciencedirect.com/topics/engineering/active-disturbance-rejection-control#:~:text=Active%20 Disturbance%20 Rejection%20Control%20

PID (Proportional-Integral-Derivative) control algorithms

PID (Proportional-Integral-Derivative) control merupakan salah satu algoritma kontrol yang sangat sering digunakan dalam sektor industri karena ketahanannya terhadap kondisi yang bervariasi dan fungsionalitasnya yang mudah dipahami. Kemampuan ini menjadikannya alat yang mudah diterapkan oleh para insinyur dalam berbagai aplikasi. PID controller bekerja dengan menerima data dari sensor, kemudian menghitung selisih antara nilai aktual dan nilai atau hasil yang diinginkan, serta menyesuaikan keluaran sistem agar sesuai dengan target tersebut. Dalam diagram blok, PID digambarkan sebagai sistem kontrol tertutup di mana informasi dari output dikembalikan ke input untuk membuat penyesuaian yang sesuai.

Dalam prakteknya, contoh penggunaan PID bisa dilihat pada sistem pengatur suhu AC. Ketika kita menetapkan suhu yang diinginkan, misalnya 25 derajat Celsius, sistem AC akan terus mengukur suhu ruangan. Jika suhu aktual ruangan tidak sesuai dengan yang diinginkan, PID controller akan menghasilkan sinyal output untuk memerintahkan sistem AC agar menyesuaikan suhu, baik dengan menurunkan atau menaikkannya. Ketika nilai yang dihasilkan tidak sesuai dengan target, kesalahan (error) ini akan diproses dalam tiga bagian utama PID: Proportional, Integral, dan Derivative.

Bagian pertama, yaitu **Proportional** (P), mengirimkan sinyal output yang proporsional dengan error yang terjadi, dengan rumus

$$P\ output = K_p \times error$$

Komponen ini bekerja untuk mendekatkan nilai keluaran sistem terhadap nilai yang diinginkan, namun tidak selalu mencapai nilai tepat. Pengaturan konstanta K_p harus dilakukan dengan hati-hati karena nilai yang terlalu besar bisa menyebabkan osilasi, sementara nilai yang terlalu kecil tidak cukup efektif.

Selanjutnya, **Integral** (I) bekerja dengan memperhitungkan akumulasi error dari waktu ke waktu, sehingga outputnya proporsional terhadap magnitude dan durasi error. Rumus untuk komponen ini adalah

$$I\ output = Ki \times \int error\, dt$$

Fungsi integral ini berperan penting dalam menghilangkan steady-state error dengan memperbaiki kesalahan yang terus-menerus terjadi dalam sistem.

Terakhir, komponen **Derivative** (D) berfungsi untuk memprediksi tren error berdasarkan tingkat perubahan error tersebut, dengan rumus

$$D\ Output = Kd \times \frac{d(Error)}{dt}$$

Komponen derivative ini membantu dalam mengurangi osilasi dan mempercepat respons sistem dengan merespons perubahan yang cepat dalam error.

Ketiga komponen P, I, dan D ini bekerja secara sinergis untuk mengatur koreksi yang dibutuhkan dalam sistem agar nilai keluaran dapat sesuai dengan target yang diinginkan. Dengan kombinasi ini, PID controller mampu mengoptimalkan performa sistem kontrol,

seperti menjaga suhu, tekanan, posisi, atau kecepatan sesuai dengan yang diharapkan, menjadikannya solusi yang kuat dan andal dalam berbagai aplikasi industri dan otomasi.

sumber:
https://www.ni.com/en/shop/labview/pid-theory-explained.html?srsltid=AfmBOorQN_8HaT68L5jx8cEXIiHdgwqq36ISFYFX232Mp42vqGTRtTV8

A* (A star) algorithm

A* (A star) algorithm adalah algoritma yang sangat efisien dan sering digunakan dalam pencarian jalur dan traversal grafik, dianggap sebagai salah satu algoritma terbaik untuk keperluan ini. Algoritma ini umumnya diterapkan pada sistem navigasi seperti Google Maps dan juga digunakan dalam pemrograman karakter non-playable (NPC) dalam gim 3D untuk menemukan jalur optimal dari satu titik ke titik lainnya. Misalnya, ketika kita ingin berpindah dari titik A ke titik Z, A* akan membantu kita menemukan jalur terpendek atau jalur yang paling efisien untuk sampai ke tujuan.

Algoritma A* bekerja dengan mendefinisikan fungsi-fungsi penting yang akan membantu dalam proses pencarian jalur. Fungsi-fungsi ini adalah fungsi f, g, dan h. Fungsi f(n) melambangkan total perkiraan biaya jalur melalui node n, yang merupakan penjumlahan dari fungsi g dan fungsi h. Fungsi g(n) melambangkan biaya atau jarak sebenarnya dari titik awal ke node n, sedangkan fungsi h(n) disebut sebagai heuristic, yaitu perkiraan biaya terpendek dari node n ke tujuan. Sehingga, rumus yang digunakan dalam A* adalah:

$$f(n)=g(n)+h(n).$$

Fungsi heuristic dalam algoritma A* sangat penting karena dapat mempengaruhi efisiensi pencarian jalur. Tergantung pada layout peta atau lingkungan yang sedang dipetakan, nilai dari fungsi heuristic ini bisa berbeda-beda. Beberapa heuristic yang umum digunakan meliputi:

- Jarak Manhattan → Digunakan untuk grid yang hanya memperbolehkan gerakan horizontal dan vertikal.
- Jarak Euclidean → Cocok untuk ruang kontinu yang memungkinkan pergerakan diagonal.
- Jarak Octile → Digunakan jika gerakan diagonal memiliki biaya yang berbeda.

Proses pencarian jalur terpendek dalam algoritma A* melibatkan beberapa langkah utama, yaitu inisialisasi dan loop utama yang berulang sampai tujuan tercapai atau tidak ada lagi jalur yang dapat dieksplorasi.

1. Inisialisasi: Pada tahap ini, kita membuat dua daftar: open list, yang berisi node-node yang akan dievaluasi, dan closed list, yang berisi node-node yang sudah dievaluasi. Node awal ditempatkan dalam open list dengan nilai awal g(0) = 0 dan f(0) = h(0)
2. Loop Utama: Selama open list tidak kosong, langkah-langkah berikut dilakukan:
    - Pilih Node Terbaik: Pilih node dengan nilai f(n) terendah dalam open list dan sebut node tersebut sebagai b. Node ini dihapus dari open list dan dipindahkan ke closed list.
    - Buat Node Suksesor: Buat semua node tetangga dari node b dan atur node b sebagai parent dari masing-masing node suksesor.
    - Evaluasi Suksesor: Untuk setiap suksesor, lakukan langkah-langkah berikut:

        - Jika suksesor adalah tujuan, hentikan pencarian karena jalur telah ditemukan.
        - Jika tidak, hitung nilai g dan h untuk suksesor tersebut menggunakan jarak seperti Manhattan, Euclidean, atau Octile tergantung pada peta yang digunakan.
        - Hitung nilai f untuk suksesor menggunakan rumus f = g + h
        - Jika ada node dengan posisi yang sama di open list dengan nilai f lebih rendah, abaikan suksesor ini.
        - Jika ada node dengan posisi yang sama di closed list dengan nilai f lebih rendah, abaikan suksesor ini juga. Jika tidak, tambahkan suksesor ke open list.
    - Ulangi: Proses ini terus diulang sampai tujuan tercapai atau open list menjadi kosong.

Algoritma A* menggabungkan pendekatan greedy search yang cepat namun kurang akurat dengan pendekatan uniform-cost search yang lebih lambat tapi lebih akurat. Kombinasi ini menjadikan A* algoritma yang efisien dan optimal untuk banyak aplikasi, seperti pencarian jalur di peta, game AI, dan berbagai aplikasi robotika. Efisiensinya sangat bergantung pada pemilihan fungsi heuristic yang tepat, sehingga dapat meminimalkan jumlah node yang perlu dieksplorasi sebelum mencapai solusi.

sumber:
https://www.geeksforgeeks.org/a-search-algorithm/
https://medium.com/@preston.elliott/a-star-algorithm-e5d801b24a68
https://www.youtube.com/watch?v=PzEWHH2v3TE